Technological Educational Institute of Crete
School of Applied Technology
Department of Informatics Engineering

Paper Title

Integrating WebRTC and X3DOM: Bridging the Gap between Communications and Graphics

Haroula Andrioti, Andreas Stamoulias, Kostas Kapetanakis, Spyros Panagiotakis, Athanasios G. Malamos

Presentation by: Dr. Athanasios G. Malamos

Introductory notes

- WebRTC is the standardized project that provides browsers and mobile applications with **Real Time Conferencing** capabilities via JavaScript APIs.

- This opens new horizons in web-based applications such as online gaming, video-conferencing, exchanging of text messages, immersive technology, etc.

- We introduce the integration of WebRTC capabilities within virtual 3D worlds and present several implementations that bridge WebRTC and X3DOM technologies.

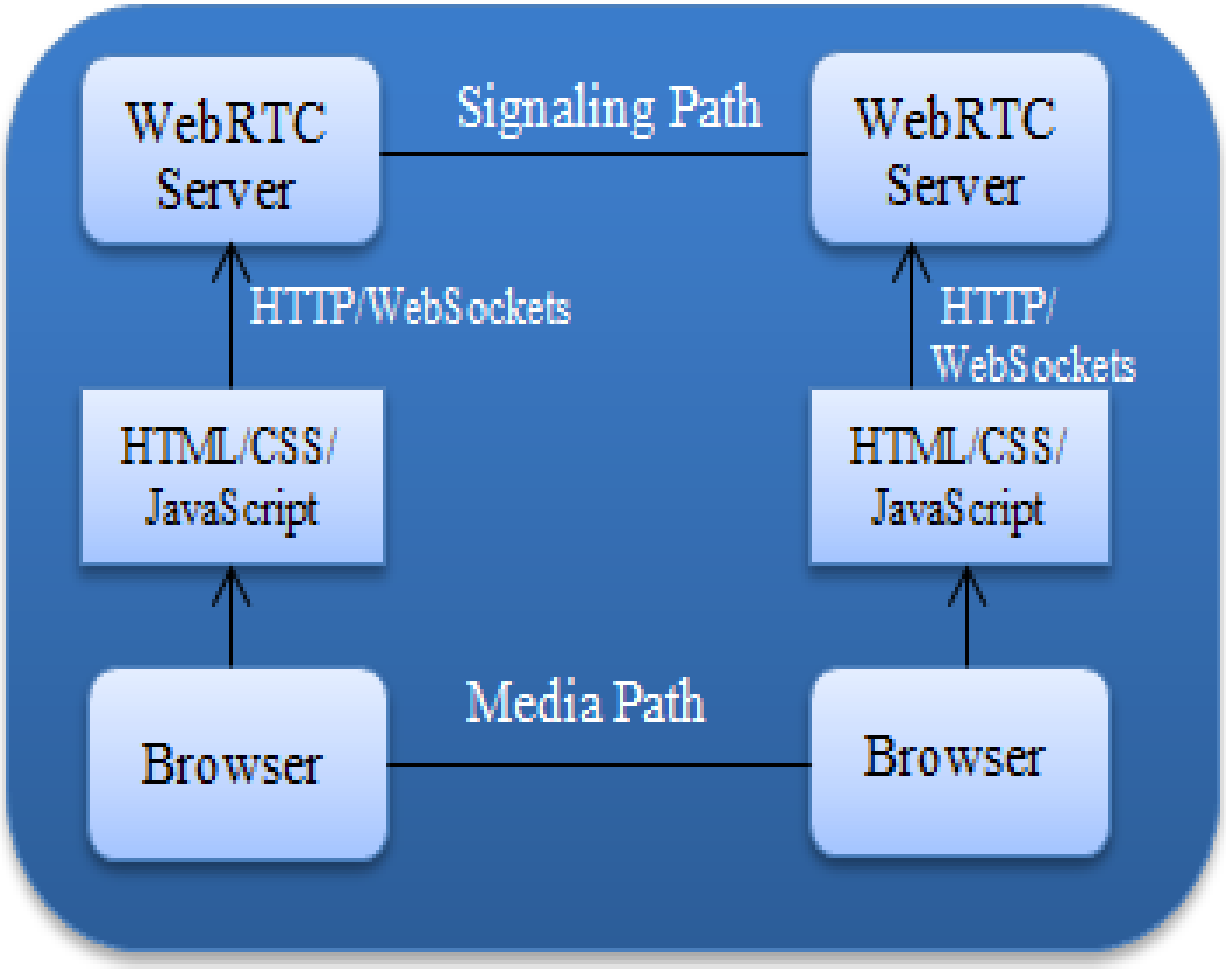
A few words about WebRTC…

The goal of WebRTC is to provide real-time communication among browsers in a P2P connection with no use of plugins.

The **Three Main APIs** of WebRTC

- **MediaStream** API
  - Obtaining audio & video. getUserMedia() method
- **RTCPeerConnection** API
  - Holds audio & video connection between peers
- **RTCDataChannel** API
  - Bidirectional communication between peers for arbitrary data

WebRTC architecture



Network Address Translation

- STUN (Session Traversal Utilities for NAT) is a client-server protocol. STUN reveals for every user its public IP address and port.
- TURN (Traversal Using Relays around NAT) is a relay server. TURN is used as a final solution if it fails previously to establish the connection using STUN server
- ICE(Interactive Connectivity Establishment) is an umbrella framework. ICE tries to find the best solution selecting between STUN and TURN. Firstly, ICE tries to establish connection between peers directly via UDP using STUN server. If this process does not succeed, ICE tries to connect using TCP. If that fails as well, then ICE uses a TURN relay server.

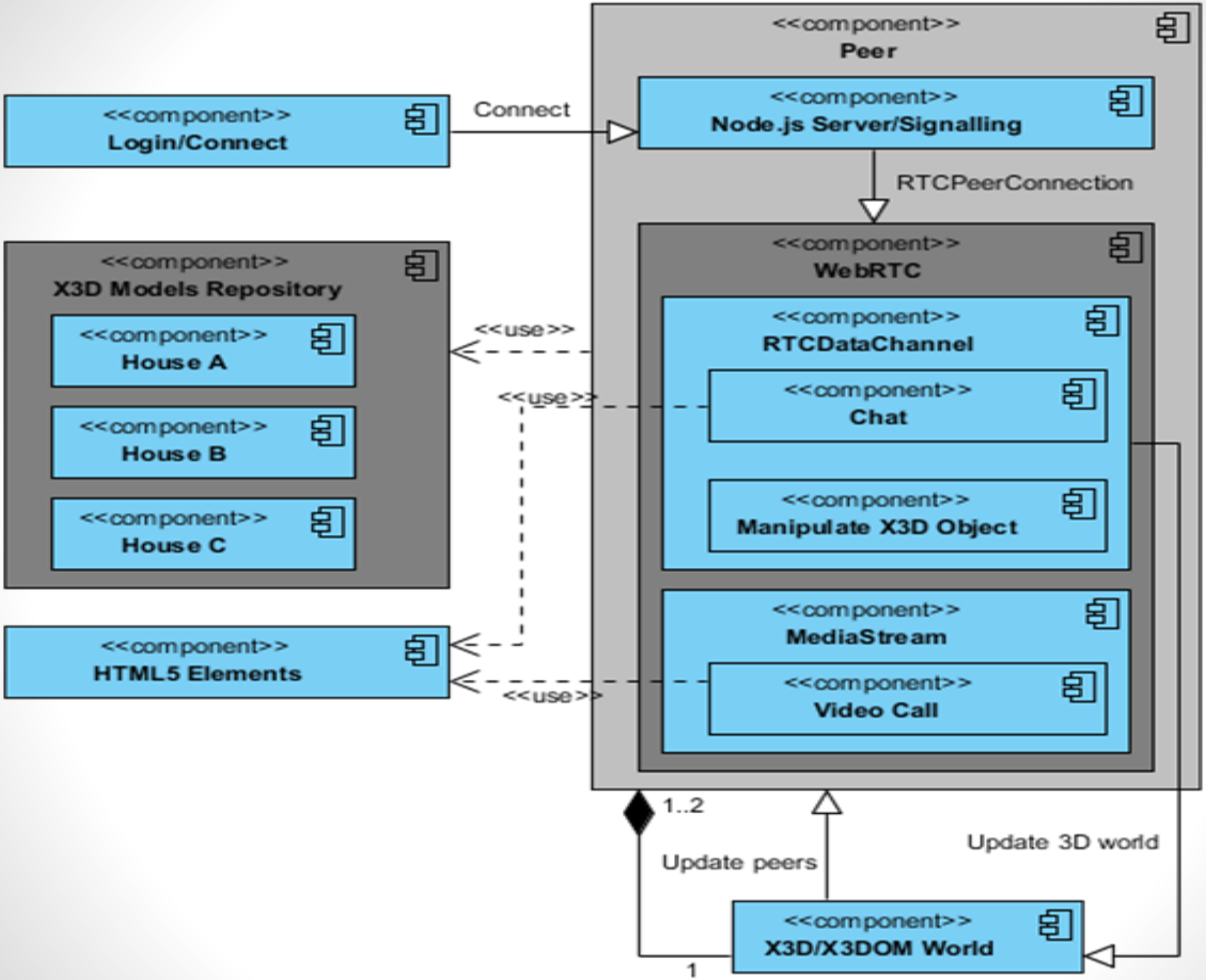
Tools that were used

In order to accomplish our applications we used:

- HTML5
- JavaScript
- Node.js
- Socket.io

First Application: 3D Collaborative Educational Online Game

- Our scope was to implement a web-based application which provides a 3D collaborative environment to exchange Video/audio and X3D objects using WebRTC technology.


<<component>>
Peer
<<component>>
Login/Connect
Connect
<<component>>
Node.js Server/Signalling
RTCPeerConnection
<<component>>
WebRTC
<<component>>
X3D Models Repository
<<component>>
House A
<<component>>
House B
<<component>>
House C
<<use>>
<<use>>
<<component>>
RTCDataChannel
<<component>>
Chat
<<component>>
Manipulate X3D Object
<<component>>
MediaStream
<<component>>
Video Call
<<component>>
HTML5 Elements
<<use>>
1..2
Update peers
Update 3D world
<<component>>
X3D/X3DOM World
1

Second application: Immersive Application

- The objective of this application concerns a 3D collaborative environment that supports real-time communication network between peers using a plugin free, peer-to-peer connection.
- **The main advantage of our application is that real-time video is projected inside the 3D world**

Our secret scope is to prove that if you want to use WebRTC inside 3D scenes……. Do it with X3D. Its smooth and efficient!

Draft architecture

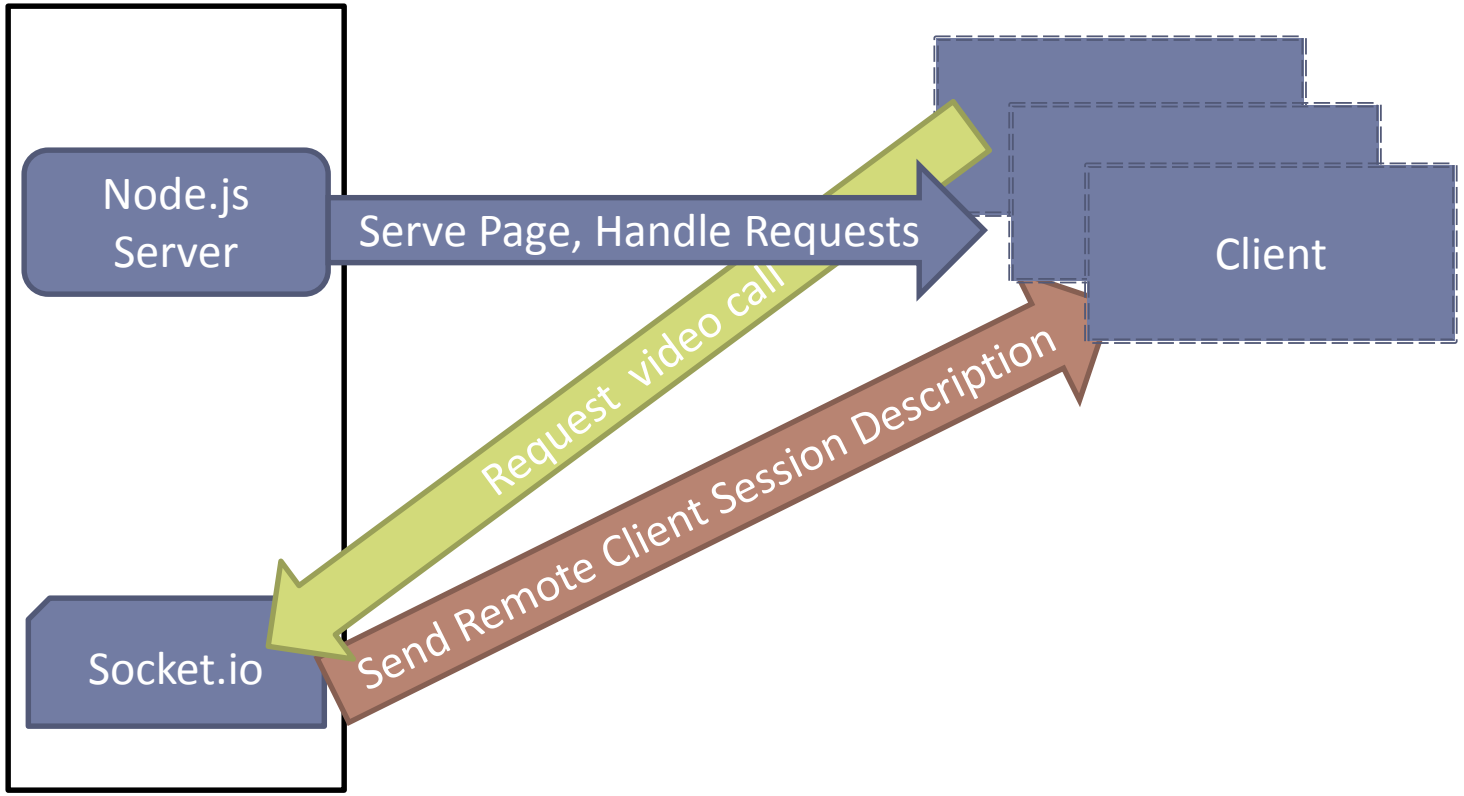
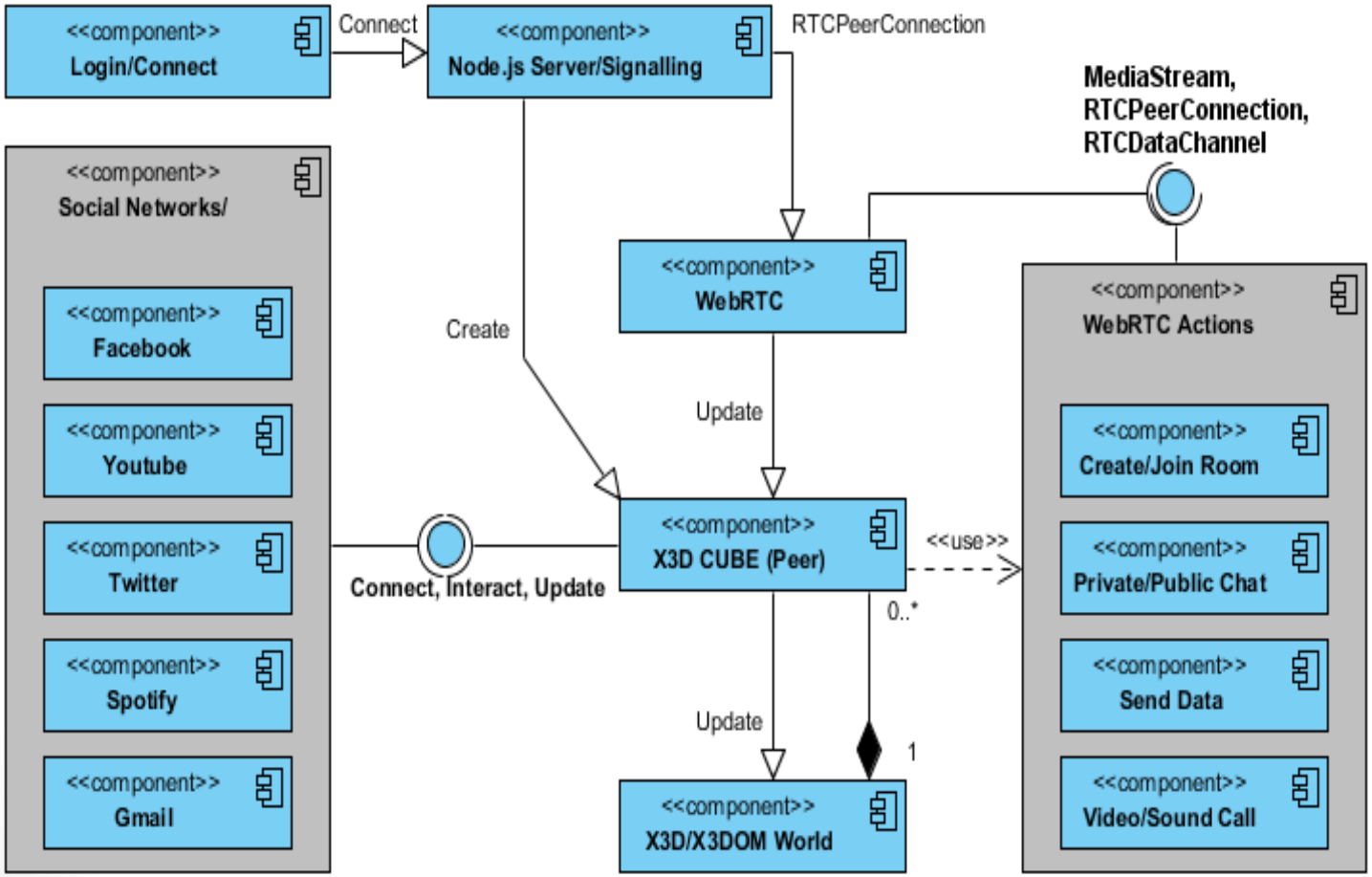


Diagram of the Immersive – Experience application



Coding examples…….

Connection 1/2

Client connects via Socket.io to Node.js Server hosted at OpenShift

Server

```
var myApp = require('http').createServer(handler);
var url = require('url');
var fs = require('fs');
var path = require("path");
var io = require('socket.io')(myApp);
var server_port = process.env.OPENSHIFT_NODEJS_PORT || 8080
var server_ip_address = process.env.OPENSHIFT_NODEJS_IP || '127.0.0.1'
myApp.listen(server_port, server_ip_address, function(){
        console.log("Listening on " + server_ip_address + ", server_port " + server_port);
});
```

Client

```
var socket = io.connect('server_ip_addres: server_port ');
```

Connection 2/2

Client utilizing the getUserMedia, attaches its own video stream to an X3D shape by appending a MovieTexture element with the stream to it's appearance element.

```
var RTCPeerConnection = window.RTCPeerConnection || window.mozRTCPeerConnection || window.webkitRTCPeerConnection;
var SessionDescription = window.RTCSessionDescription || window.mozRTCSessionDescription || window.webkitRTCSessionDescription;
var pc = new RTCPeerConnection({"iceServers": []});
navigator.getUserMedia = navigator.getUserMedia || navigator.webkitGetUserMedia || navigator.mozGetUserMedia || navigator.msGetUserMedia;
var constraints={video:true, audio:true};
navigator.getUserMedia(constraints, function (stream) {
    var localvideo = document.getElementById("localVideo");
    localvideo.src = window.URL.createObjectURL(stream);
    var movieTextureNode = document.createElement('MovieTexture');
    movieTextureNode.setAttribute("url", '"' + localvideo.src + '"');
    if(document.getElementById("video_"+myPanel)){
    document.getElementById("video_"+myPanel).appendChild(movieTextureNode);
    document.getElementById("frame_"+myPanel)._x3domNode._cf.material.node._xmlNode.setAttribute("diffuseColor", "0 1 0");
}};
```

Video Call-Request

Client create an RTCPeerConnection offer by adding it's stream and setting the local session description and then send the video call request to the server through the open socket with the session description data.

```
pc.addStream(stream);
pc.createOffer( function (description) {
        pc.setLocalDescription(new SessionDescription(description));
        socket.emit("video call", {type: "offer", "description": description});
});
```

- Server broadcast the client offer request to the other peer

```
socket.on('video call', function(data){
        socket.broadcast.emit('video call', data);
});
```

Video Call-Response

- Client gets the remote offer request, set the remote session description data and then create an answer with it's own local session description data.
- Client send the video call answer to the server through the open socket with the session description data.
- Server broadcast the client answer to the other peer and start the video call

```
socket.on("video call", function(data) {
    switch(data.type){

        case "iceCandidate":
          RTCIceCandidate(data.candidate);
          pc.addIceCandidate(new RTCIceCandidate(data.candidate));
        break;

        case "offer":
          pc.setRemoteDescription(new SessionDescription(data.description));
          pc.createAnswer(function(description) {
          pc.setLocalDescription(new SessionDescription(description));
          socket.emit("video call", {type: "answer", "description": description});
        });
        break;

        case "answer":
          pc.setRemoteDescription(new SessionDescription(data.description));
        break;
    }});
```

Immersive application as shown online

DEMO

Conclusions & Future Work

- We presented the integration of WebRTC and X3DOM technologies
- In our developments, we enabled 3D collaborative environments in which the first one is created for online gaming capabilities and the second one to support video chat with multiple users, text messaging, and being able to log in social media accounts such as email, Facebook, twitter, etc.
- We strongly believe in the synergy between WebRTC and X3DOM that may widen the range of promising web applications and services.
- Other technological fields such as online gaming, e-learning and e-health can benefit from the peer-to-peer and server-free nature of WebRTC.

Thank you!

20th International Conference on 3D Web Technology
June 18-21 –Heraklion,Crete,Greece